AFRL-IF-RS-TR-2004-107
Final Technical Report
April 2004

RESEARCH ON THE KNOWLEDGE PLANE: DISTRIBUTED LEARNING AND REASONING ABOUT NETWORK OUTAGES

Institute for the Study of Learning and Expertise

Sponsored by
Defense Advanced Research Projects Agency
DARPA Order No. Q182

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE
ROME RESEARCH SITE
ROME, NEW YORK

STINFO FINAL REPORT

This report has been reviewed by the Air Force Research Laboratory, Information Directorate, Public Affairs Office (IFOIPA) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

AFRL-IF-RS-TR-2004-107 has been reviewed and is approved for publication

APPROVED: /s/

ALAN J. AKINS
Project Engineer

FOR THE DIRECTOR: /s/

WARREN H. DEBANY, JR., Technical Advisor
Information Grid Division
Information Directorate

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	APRIL 2004	Final Jun 03 – Oct 03

4. TITLE AND SUBTITLE
RESEARCH ON THE KNOWLEDGE PLANE: DISTRIBUTED LEARNING AND REASONING ABOUT NETWORK OUTAGES

5. FUNDING NUMBERS
C - F30602-03-2-0198
PE - 61101E
PR - Q182
TA - GA
WU - 01

6. AUTHOR(S)
Pat Langley and Stephen D. Bay

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Institute for the Study of Learning and Expertise
2164 Staunton Court
Palo Alto California 94306-1438

8. PERFORMING ORGANIZATION REPORT NUMBER
N/A

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Defense Advanced Research Projects Agency
3701 North Fairfax Drive
Arlington Virginia 22203-1714

AFRL/IFGA
525 Brooks Road
Rome New York 13441-4505

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
AFRL-IF-RS-TR-2004-107

11. SUPPLEMENTARY NOTES
AFRL Project Engineer: Alan J. Akins/IFGA/(315) 330-1869/ Alan.Akins@rl.af.mil

12a. DISTRIBUTION / AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*
This report summarizes experiences developing a demonstration system to perform distributed detection of network failures. A simple system was developed based on probabilistically inferring failed nodes from observations on requested traffic and propagated error messages. Simulations verified that network failures could be automatically detected with limited memory and communication overhead.

14. SUBJECT TERMS
Knowledge Plane, Network Failure Detection, Network Fault Reasoning, Probabilistic Model, Distributed Diagnosis, Network Simulation, Cyber Defense

15. NUMBER OF PAGES
12

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Table of Contents

List of Figures

1. Summary

We have been carrying out research on the task of representing, reasoning and learning about network interruptions. We have developed a distributed system whereby each node in the network monitors itself in real-time and develops and model of its normal traffic behavior. When a connection error occurs, the originating node sends an error report to a limited set of relevant nodes. Each node tracks these error messages that are passed through the network and in conjunction with its internal model of the network, infers the likely location of the network interruption with a probabilistic model.

To test our system, we have developed a network simulator that automatically creates arbitrary sized networks and generates traffic according to a specified probability distribution. The number of end nodes per major hub is a parameter that can be set from 10 to 1000. We injected faults into this network and monitored the ability of individual nodes to detect the fault. From experimental runs with the simulator we have observed the following.

- Local diagnosis of distant connection problem is feasible.
- The communication overhead to send error messages is small compared with regular traffic.
- Nodes in the network do not have equal ability to infer problems correctly. The nodes tend to "specialize" and are better able to detect errors in nodes with which they typically exchange more traffic than average.
- The memory overhead to implement this system is large but does not grow as the simulation runs. The cost is also distributed across the network.

We believe future research on this topic should concentrate on extending the representational power of the models used to represent traffic flows with the goal of detecting a wider variety of connection problems.

2. Introduction

In this project, we created a simple demonstration system to test the feasibility of learning in the presence of distributed information (as would be the case for a knowledge-plane construct). Specifically, we examined the following problem.

> Given a network of nodes that can monitor themselves and communicate, build a distributed system that automatically detects connection errors and propagates this information as needed.

We believe a distributed system is necessary and even advantageous in many respects. Although a centralized solution is possible, it is less desirable because it requires communication with a central node which may not be accessible if there are network problems. Furthermore a

distributed system will spread out the cost of implementation. From this study we hope to achieve three things.

1. Demonstrate that distributed diagnosis is feasible.
2. Gain a qualitative understanding of issues involved with construction of such a system.
3. Identify open problems and tasks for future research.

2.1 Current Practice

A common method of identifying failed network connections is to use the tool traceroute which tracks the path that packets take to the destination. Traceroute works well but has some important limitations:

1. Traceroute cannot detect intermittent problems. It only determines if a connection can be made right now.
2. Not all routers provide the information that traceroute needs. E.g., some fail to observe "time to live" or do not respond at all (causing traceroute to time out)
3. Traceroute does not tell if a specific service is available but only if the nodes respond to ICMP.
4. If the destination node is a popular site (e.g., CNN), multiple requesting sites all performing traceroute will result in heavy bandwidth.
5. There is no extension path from traceroute to more sophisticated models that would allow reasoning about other network characteristics. Thus, it would be difficult to extend a traceroute system to serve requests like "Schedule my download of file X when the connection to node Y is fastest".

3. Methods, Assumptions, and Procedures

We assume that the network is instrumented with *think-points* (TP) [2] at major hubs (autonomous systems) and end user stations. Each TP is self contained reasoning unit that can observe network traffic through that point. Furthermore, when a node makes a traffic request that fails to reach its destination (no return information) each TP can generate an error message that is propagated to other nodes in the network.

In general terms, our framework for detecting network failures works as follows:

- Each TP learns a model *T* of its traffic.
- Each TP learns a model *E* of recent failed requests.

- Given *T* and *E* the TP infers the state of the network.

In this report, we will assume that the only failure mode is a site going completely offline, in which case it can neither receive nor forward messages. Each TP develops a hierarchical model of its own traffic and errors. When a connection error occurs the source node propagates an error message to TPs along the same routing path as the initial request. By examining the traffic and error each node can reason about the network state.

We discuss an example which shows how the system might work at a general level and then we describe the components in greater detail.

3.1 Example Problem

We illustrate our ideas with an example problem. Consider a system composed of four autonomous systems (AS): A, B, C, and D and shown in Figure 2. Each AS is connected to a variety of lower level nodes (e.g., a1, a2, a3 etc); the lower nodes for B and D are not shown for clarity reasons. We will use uppercase letters for autonomous systems and lowercase for end nodes. Each AS may have many associated end nodes. Finally, each AS has an associated TP which can observe and reason from its observations.

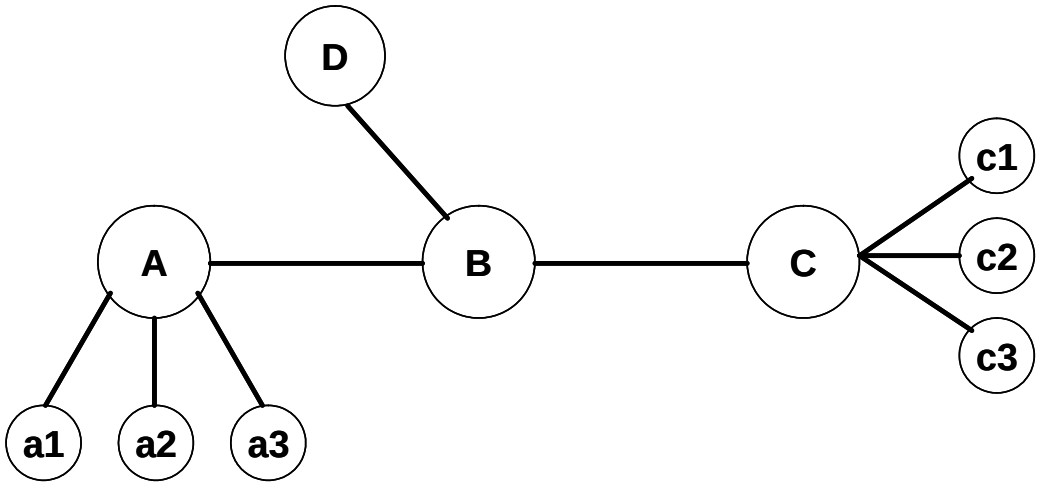

Figure 1: Example Network

Suppose the node a1 makes a request of node c1, which under working conditions would be routed along the path {a1, A, B, C, c1}. However, there is an error in the network and a1 receives no response from c1. Once a1 determines that its attempt to connect to c1 has failed (i.e., no response after a fixed time limit), the TP at node a1 then generates an error report and sends it along the same route as the request. Note that both the traffic request and error message will be forwarded along the routing path until it encounters a failed node. Each node on the path

takes note of a1's trouble report and updates its own statistics about the nodes that have connection errors.

The network outage could be at several locations along the path from a1 to c1, and possible candidates are a1, A, B, C, or c1. The TP at a1 can infer based on T and E the likely location of the fault with the following type of reasoning.

- If a1 does a lot of traffic with c2, and there are no failures a1-c2 then a1 can conclude that the outage is at c1. To reach c2 the path {a1, A, B, C} must be operational and thus the error has to be at c1.

- If a1 also experiences many other failed requests to nodes such as B or D then the error may be closer in the network (e.g., B).

- If a1 can not communicate with any other node, then the error is probably with a1 itself.

The TP at A also repeats this analysis process. However, A processes much more traffic as it routes messages from a2 and a3 in addition to a1. Thus A should be able to better distinguish the failure cause since it has more observations from which to make an inference. Likewise, the other nodes that receive the error message (i.e., B and D) also attempt to infer the error location.

In the next two sections, we discuss how the TPs can model their traffic and errors, and how they can use this information to reason probabilistically about the possible faults.

3.2 Traffic and Error Models

We model the traffic and errors with a two-level hierarchy. Each node keeps track of the traffic going to any destination (end node) and the corresponding AS. We assume that every message sent has an address that can be decoded into an AS and end node.

The TPs keep track of the traffic to the various destinations with an exponentially weighted moving average:

$$T(n,t) = T(n,t-1)(\lambda) + M(n,t)(1-\lambda) \qquad (1)$$

The function $T(n,t)$ represents the estimated traffic demand to node n at time t and is a weighted sum of the previous traffic estimate plus the messages sent to n at time t. The variable λ is a decay constant and should be set between zero and one. For our experiments, 0.9 was used as the value of λ. The exponentially weighted moving average has the advantage of being extremely simple to update and requires recording only one number per destination.

We store the traffic counts for different destinations in a hash structure which allows fast access and retrieval of counts. However, as a node makes requests to different destinations, the number of counters needed may grow. To prevent the size of the data structure necessary to store the

traffic numbers growing too large, addresses with small scores are deleted from the structure. Thus for example, a rare traffic destination would be quickly removed from the data structure.

3.3 Reasoning about errors

We treat the reasoning problem as a classification problem where the classes correspond to the possible failure states of the network. Each TP maintains a structure of traffic and errors, and from these can infer potential outages in the network. The TP then selects the class that is most probable given the observations.

We use a naive Bayesian classifier to perform the inference [1]. The naive Bayesian classifier is based on Bayes rule which states

$$P(c/x) = \frac{P(x/c)P(c)}{P(x)} \tag{2}$$

i.e., the probability of a class *c* given an observation vector *x* is proportional to the probability of *x* given *c*. The term *x* corresponds to the requests that were either successful or failures and can be computed from *T* and *E*. The naive Bayesian classifier makes the assumption that all observations are independent given the class and thus

$$P(c/x) = \frac{\Pi_i P(x_i/c)P(c)}{P(x)} \tag{3}$$

For each AS and its associated end nodes we consider two cases. The first is that the AS itself is broken. If a node is broken we expect that with probability δ a traffic request will result in an error, and 1 – δ no error. Note that δ is not set to zero because the exponentially weighted moving average incorporates information from the recent history of traffic and thus even if the node is now working correctly, the estimate may include past errors. For a hub failure we define

$$\prod_i P(x_i/c_{hub}) = \delta^e * (1-\delta)^{(t-e)} \tag{4}$$

Where *t* and *e* represent the total counts from the traffic models for the hub of interest.

The second case we consider is that some individual end nodes are broken but the AS is fine. We model the joint probability of the individual end nodes generating the observations as the product of the probabilities for those individual nodes that are hypothesized to be working or experiencing a failure.

$$\prod_i P(x_i/c_{nodes}) = \prod_{\in working} \prod_i \delta^{e_i} * (1-\delta)^{(t_i-e_i)} \prod_{\in failed} \prod_i \gamma^{e_i} * (1-\gamma)^{(t_i-e_i)} \tag{5}$$

Where t_i and e_i represent the total counts to each end node, and γ is the probability of observing an error for a working node.

4. Results and Discussion

In this section, we discuss experiments with our system in detecting network errors. We note that we are most concerned with discovering qualitative behaviors of the system and not detailed quantitive comparisons.

We implemented our system in C++ and ran experiments under UNIX. We generated random networks that varied the number of AS and end nodes. Additionally, traffic was generated by randomly selecting the origin and destination according to a non-uniform distribution.

We experimented with several fixed structures as in Figure 1, as well as randomly generated networks where we inserted faults. The network ranges in size from a small system with 4 AS and approximately 10 end nodes per AS to 100 AS each with 100 end nodes.

We present here the results of a simulation run on Figure 1 where a network error was injected at node C. Figure 2 shows the inferred errors for the TP located at nodes B, b3 and A. From the graph, we make the following conclusions.

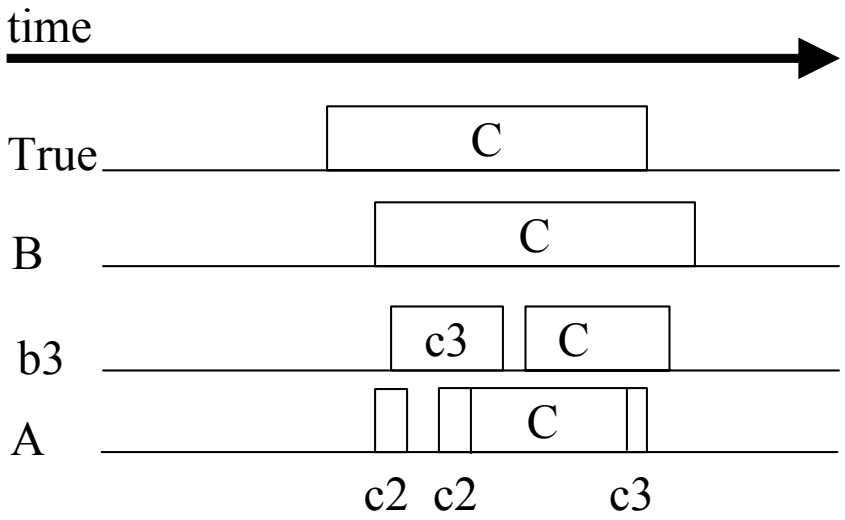

Figure 2: Simulation trace with a network error. The timelines show the inferred error by the TPs at points B, b3, and A.

- All nodes were able to detect a failure in node C.
- There is a lag in detecting the failure. This is a direct result of using a moving average to store traffic and error information.

- Each TP has good statistics for nodes "close" to it and therefore can more accurately diagnose problems. For instance, the TP at B performed much better than b3 or A.

We ran several simulations varying the size of the network, as well as the errors that could occur. In all cases the extra overhead for sending error measurements was small and typically less than 10% of the total traffic. The overhead for storing the memory needed for the traffic and error models was large but fixed and did not grow as time passes. However, the cost is mitigated by the fact that the cost is distributed across many nodes in the network.

5. Limitations and Future Research

Our experiments demonstrated that distributed learning of network failures was feasible. In this section, we discuss some of the approach's limitations and directions for future research.

- *Incorporating state information.* Our representation in section 3.1 was stateless and did not consider the time ordering of examples. The stateless representation means that it could be represented efficiently, but the drawback was that there was some lag in identifying failures. Storing the complete time history of traffic and errors could alleviate this problem but obviously requires greater memory. An important task then would be to selectively remember certain facts.
- *Developing more sophisticated models of internet performance and classes of failures.* Our models only tracked successful and failed connections between nodes in the network. There would be much benefit to tracking other characteristics such as latency or bandwidth. This would require more sophisticated models, but would also allow inference on more complex queries.
- *Extending the available actions to the TP.* Currently, the TPs can only propagate error messages. Other actions might request another node's diagnosis of the problem.
- *Incorporating knowledge of topology into reasoning processes.* We assumed nodes could only tell which AS and end-nodes were associated, but not how the AS's are connected. Knowledge of the topology would let more complex hypothesis about network failures be considered.

6. Conclusions

We constructed a demonstration system and showed that distributed diagnosis of network faults was feasible. In the distributed system, each end node becomes a detector of network faults and can signal when an error has occurred. Experiments with a simulator showed that faults could be detected with a limited overhead for error messages and memory. Future work should focus on the models used by each node to model the traffic and errors that occur.

References

[1] P. Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 223-228, 1992.

[2} C. Partridge. Thoughts on the structure of the knowledge plane. http://www.isi.edu/~braden/know-plane/, 2003.